JSiSE Research Report
vol.34,no.2(2019-7)

写経型プログラミング学習における無作為修正者の検出

江籠育朗[*1], 内田眞司[*2], 上野秀剛[*2], 大野優[*3], 河合和久[*4]
*1 奈良工業高等専門学校専攻科　システム創成工学専攻　情報システムコース
*2 奈良工業高等専門学校　情報工学科
*3 ゾーホージャパン株式会社
*4 豊橋技術科学大学　情報・知能工学系

Detection of Random Correction in the 'SHAKYO'-Style Learning of Computer Programming

Ikuro Ego[*1], Shinji Uchida[*2], Hideaki Uwano[*2], Yu Ohno[*3], Kazuhisa Kawai[*4]
*1 National Institute of Technology, Nara College, Faculty of Advanced Engineering
*2 National Institute of Technology, Nara College, Department of Information Engineering
*3 ZOHO Japan Corporation
*4 Toyohashi University of Technology, Department of Computer Science and Engineering

"SHAKYO'-Style learning of computer novice is one of the learning methods for programming beginners. This method is the learning of computer programming by mimicking input according to sample programs, running them and ensuring their outcome. In programming education, it is improve educational effect to detect student who actions the "random correction". Random correction is as an action that source code correction without understanding the exercise contents. Our purpose is to classify the students who is random correction in 'SHAKYO'-Style learning on the Online Judge System(OJS). OJS enable for learners to submit and execute programs online. OJS store the past records of code that user submit the server as Snapshot. In this paper, we propose 6 metrics which is characteristic of behavior from OJS snapshot and verify the usefulness.

キーワード: プログラミング教育，写経型プログラミング学習，OJS，無作為修正者

1. はじめに

プログラミング教育において初学者の学習方法のひとつに写経型プログラミング学習（以降，写経型学習と呼ぶ.）がある．写経型学習とは教員または教材により提示されたサンプルプログラムを学習者が模倣して入力し，コンパイル・実行・結果の確認を行う一連の流れの学習法である(1)．写経型学習により個々の宣言的知識を整理，統合して有機的に結び付け，利用可能な知識として習得することが可能となる．

一方でプログラミング教育は多数の受講生に対して少人数の教員が実施することが多い．そのため受講生の理解状況や作業の様子を教員が把握することは容易ではない．教員への負担が増えるだけでなく，理解が不十分な学生に対し指導が行き届かない場合もある．このような状況は学習者に対して学習意欲の低下を招くことも懸念される．

近年，プログラミング教育環境に Online Judge System(OJS)を導入している事例が報告されている.OJS はユーザから提出されたプログラムをコンパイル・実行し，正誤判定データや検証器を用いて，正確さと性能を自動評価し，フィードバックするシステムである(5)．OJS を教育現場で導入することで，教員の教育コスト削減や学生のプログラミング学習の効率化に繋がることが報告されている(6)．学生たちに単純に問題を解かせてもプログラミング能力が身につかな

い, という導入プログラミング演習の問題点から, OJS を用いて改善しようという試みもある[7]. OJS には提出されたコードをスナップショットとして記録される. スナップショットの履歴を分析することで理解が不十分な学習者を検出できれば，教員はその学習者に対して別途指導が容易となる.

本研究では初学者に対するプログラミング教育現場を対象とし，写経型学習において無作為修正者を OJS のスナップショットの履歴から検出することを目的とする．無作為修正者とは，OJS に提出したコードの修正を迫られた時，一般的に間違えた箇所をサンプルコード（写した元のコード）と見比べることで修正するところを，比べることなく修正する学習者を指す．このような振舞いを繰り返すことはコードの内容を理解できず，プログラムの正しい挙動を理解することを妨げてしまうので，本来想定している学習到達目標に到達できない恐れがある．このような無作為修正者は学習意欲が低く，特に指導が必要だと考える.

本稿では無作為修正者を検出するための手法として先行研究を基に OJS のスナップショットから無作為修正者の行動の特徴となる 6 つのメトリクスを提案し，その有用性を検証する.

2. 先行研究

2.1 写経型学習におけるつまずきの抽出とその改善

岡本らは写経型学習について定義し，これを用いた演習過程全体を一連の過程として，どのようなつまずきを学習者は抱えているのかを抽出した．その結果,「作業の自立性」に係るつまずきと,「作業を介した理解」に係るつまずきの 2 つに類型化し，さらに，認知的負荷理論を用いて「本質的な認知的負荷を伴う学習過程によるつまずき」と「非本質的な認知的負荷を伴う学習過程によるつまずき」に分けた[1]. また, 抽出・類型化したつまずきから教材を改善し，その有効性を検証した[2]．岡本らが定義したつまずきは，ある課題に対しエラーが発生した時，自ら修正することができず学習そのものが停滞してしまう要因と捉えることができる.

先行研究では学習者が一般的にどのようなつまずきを持っているかを定性的な手法で抽出した．つまずきを学習者の行動から定量的にとらえることができれば，どの学習者がつまずきを持っているかを検出することが可能と考える．本研究はこの類型化された中で「作業の自立性」に係るつまずきを改善するものになると考える．本研究で検出する無作為修正者は写経型学習なので正解となるコードは既にある．それを見比べることができるにも関わらず，十分に見比べない，あるいは全く見ることもなく適当に修正するので,「作業の自立性」が損なわれていると考える.

2.2 スナップショットに基づいた無作為修正の検出

大野らは奈良高専のプログラム演習受講者を対象に無作為修正者の検出を試みた[3]．具体的には以下に示す観点から行動の特徴となるような 4 つのメトリクスを提案し，各課題のテストケースの正答数から Score を算出し，その相関を求めることでその有用性を検証した.

- 同じ行を頻繁に修正する
- 一定時間の修正・コンパイル回数が多い
- 1 回あたりの修正量が小さい

先行研究と本研究の違いは分析対象のソースコードの作成する過程である演習方法にある．大野らはプログラム作成演習を対象としていた．すなわち提示されたプログラムの仕様をもとに学習者が自らコードを作成する演習方法であった（以降，課題型学習と呼ぶ.）. つまり，本研究が対象としている写経型学習のように正解となるコードは一意ではない.

写経型学習，課題型学習の双方においても無作為修正者が見受けられることから，この 2 つの学習における無作為修正者の行動の特徴は一致する範囲があると考えられる．そのため，大野らが提案した 4 つのメトリクスを用いて写経型学習における無作為修正者を検出できると考える.

以上から，本稿ではこの 4 つのメトリクスは写経型学習においても有用となるか検証する.

3. 実験

3.1 対象とするデータ

本研究で対象とするデータは平成 29 年度に奈良工業高等専門学校情報工学科の 2 年生に対して開講されたプログラミング I において受講した学生が提出した

ソースコードを対象とする．本講義は反転授業を取り入れており，自学自習として教員から配布された資料に記載されたコードを写経型学習として受講生は学習していた(4)．講義時間の演習においては課題型学習が課されていた．両学習ともに Learning Management System(LMS)の 1 つである Moodle に OJS 機能を提供する CodeRunner for Moodle を利用しており，ここに蓄積されたスナップショットを回収・分析する．

本稿では 7 課題，のべ 136 人の提出したスナップショットを対象に分析を行う．

3.2 Score(t)の算出

OJS では学習者が提出したコードをコンパイル・実行して課題の作成者が予め準備したテストケースの結果と一致するかを比べることで判定する．課題型学習ではメトリクスとの相関をとる Score はこのテストケースを正答した割合を用いていた．課題を提出し始めてから t 分以内でのテストケースにおける最高点数をであった．しかし，本研究で対象とする環境ではこの写経型学習の課題におけるテストケースが少ないことが多く，100%か 0%のどちらかの場合もあった．そのため，このように極端であれば有意な相関が取れないと考えたため，本研究では新たな Score を定義する．

CodeRunner for Moodle では各課題において，提出された際の不正解になった回数から成績を算出している．これは正解したときに与えられるもので，1 回の提出のみで正解した場合 1.0 点となる．そこから提出したコードが不正解になる度に 0.1 ずつ引かれ，最低 0 点となる．本研究ではこの成績を課題 Score(t)と定義し，t 分時におけるスコアを算出する．

3.3 メトリクス

メトリクスは以下の 4 つを用いる．なお，本研究では写経型学習取り組んでいる最中の識別を目的としているため，t 分間で提出されたスナップショットに対して各メトリクスを算出する．

Freq(t):修正行の偏り

Revs(t):t 分間の提出回数

DiffLine(s,t):s 行以下の修正が続いた回数

AveDiff(t):1 回あたりの平均修正行数

Freq(t)は t 分間に提出されたコードで，どの行に偏って修正しているのかを表す．無作為修正者は写し間違えた箇所を真剣に見比べることなく，自分の思った間違いの箇所を闇雲に修正することが考えられるので，ある行に集中して修正していると考えられる．Freq(t)は修正された回数の分散を用いる．修正行に偏りがあれば分散は大きくなるので，無作為修正者は Freq(t)が大きくなると考えられる．

Revs(t)は t 分間に提出されたスナップショットの回数を表す．無作為修正者は短時間で適当な修正を行い，それを何度も OJS に提出するので，この値が大きくなると考えられる．

DiffLine(s,t)は t 分間で s 行以下の修正が続いた回数を表す．無作為修正者は同じ行を正解であるソースコードと見比べることなく繰り返し修正し，提出をするので，s が小さければ DiffLine(s,t)の値は大きくなると考えられる．本研究では s=1 の場合を計算する．

AveDiff(t)は t 分間に提出されたそれぞれのコードの修正行数を算出し，その平均を求める．無作為修正者は同じ行を何度も修正するので，1 回あたりの修正行数は小さい．そのため，AveDiff(t)は無作為修正者であるならば値が小さくなると考えられる．

上記 4 つのメトリクスは先行研究で提案されたものであるが，これらに加えて写経型学習のみの特徴として考えられる 2 つの新たなメトリクスを用いる．

Syntax(t):シンタックスエラーの回数

Test(t):テストケースエラーの回数

これは，前述の課題 Score(t)だと提出回数である Revs(t)に相関があるのは自明なため，Revs(t)のみでは不適当だと判断したためである．CodeRunner では結果がシンタックスエラーかテストケースエラーかはスナップショットに記録されている．写経型学習で提示される課題は一般的に複雑な処理ではなく，基礎的な文法等を学ぶための簡単な処理になるので単純な写し間違い（文法エラー）が多くなると考えられる．そのため，無作為修正者はシンタックスエラーを繰り返し起こすと考えられる．

3.4 算出方法

メトリクスを以下の手順で算出する．

[Step1]スナップショット群から t 分間における

表 1.Freq(t), Revs(t), DiffLine(1,t), AveDiff(t)と Score(t)の相関

t	Freq(t)		Revs(t)		DiffLine(1,t)		Avediff(t)	
	相関係数	p 値	相関係数	p 値	相関係数	p 値	相関係数	p 値
1	0.245	0.008	-0.793	0.000	-0.134	0.154	0.168	0.072
2	0.287	0.001	-0.896	0.000	-0.259	0.003	0.121	0.177
3	0.249	0.004	-0.929	0.000	-0.322	0.000	0.148	0.091
4	0.272	0.001	-0.953	0.000	-0.374	0.000	0.164	0.058
5	0.310	0.000	-0.966	0.000	-0.430	0.000	0.165	0.056
6	0.316	0.000	-0.970	0.000	-0.428	0.000	0.163	0.060
7	0.324	0.000	-0.970	0.000	-0.427	0.000	0.151	0.079
8	0.326	0.000	-0.919	0.000	-0.371	0.000	0.155	0.073
9	0.331	0.000	-0.973	0.000	-0.426	0.000	0.147	0.089
10	0.326	0.000	-0.919	0.000	-0.371	0.000	0.155	0.073

メトリクスを算出する．Freq(t)，Revs(t)，DiffLine(s,t)，AveDiff(t)の算出方法は先行研究を基にした[2]．

[Step2]同様のスナップショット群からt分間における Syntax(t), Test(t)を算出する．与えられた t 分以内にシンタックスエラーおよびテストケースエラーになった回数を課題別，学習者別に求めた．

[Step3]Score(t)を算出する．回収したスナップショット群では正解するまでは 0 点として扱われるので，各提出で Score(t)を算出して t 分ごとにまとめる．なお，Score(t)が満点の学習者，つまり一度の提出で正解したものは該当課題について問題なく学習しているとして除外した．

[Step4]算出した各メトリクスと Score(t)との相関係数および無相関検定を算出する．

本研究では t=1 から 1 分毎に区切って t=10 まで算出した．これは，対象とする写経型学習では教科書のサンプルプログラムのような基本的な処理を含む課題が多かった．したがって修正に要する時間が総じて短かったためである．本研究で分析したデータ群の場合，10 分以内で解答を終了している学習者は 90%を超えていたため，それ以降も解答を続ける学習者は無作為修正でない別のつまずきがあると考えられるが，本実験には関係ないとして，無視する．

4. 結果と考察

先行研究で提案された 4 つのメトリクスの Score(t)との相関係数と無相関検定の結果を表 1 に示す．また，本研究で提案した 2 つのメトリクスの相関係数と無相関検定の結果を表 2 に示す．4.1 以降，全ての t において有意な相関($p<0.01$)が見られなかった AveDiff(t)を除いた各メトリクスと Score(t)との相関関係について考察する．

表 2.Syntax(t), Test(t)と Score(t)との相関

t	Syntax(t)		Test(t)	
	相関係数	p 値	相関係数	p 値
1	-0.659	0.000	-0.267	0.004
2	-0.723	0.000	-0.343	0.000
3	-0.783	0.000	-0.257	0.003
4	-0.823	0.000	-0.229	0.008
5	-0.863	0.000	-0.161	0.063
6	-0.882	0.000	-0.145	0.093
7	-0.880	0.000	-0.171	0.046
8	-0.866	0.000	-0.109	0.210
9	-0.877	0.000	-0.156	0.071
10	-0.866	0.000	-0.109	0.210

4.1 Freq(t)

Freq(t)については弱い正の相関で，全ての t において有意な相関が見られた($p<0.01$). t が大きくなるにつれて相関係数が大きくなる傾向にある．つまり，Score(t)が高い学生ほど修正行数に偏りがあることを示している．無作為修正者は総じて Score(t)が低いと考えられるので，この結果は考えられた仮説とは反対の性質を示している．これは，Score(t)が高い学習者は少ない修正回数で終了していて，1 行だけ修正して正解したような場合だと偏りは大きくなるからだと考えられる．この結果は先行研究での課題型学習においても同様であった．

4.2 DiffLine(1,t)

DiffLine(1,t)は弱い負の相関で，t=1 を除くその他の t において有意な相関が見られた($p<0.01$). これは，1 行のみの修正を何度も行っている学習者ほど Score(t)が低い，つまり無作為修正者としての傾向があることが示唆される．また先行研究より相関係数が高い傾向にあった．その理由としては，写経型学習だと正解となるコードがあるため，Score(t)が高い学習者となる学習に意欲がある学習者や既に十分な理解をしている学習者は見比べながら複数の行を修正しているためと考えられる．

4.3 Revs(t)

Revs(t)は強い負の相関を示した．また，すべての t において有意な相関が見られた($p<0.01$). 3.2 で述べた通り Score(t)は Revs(t)と関係が強いため相関が強くなるのは当然の結果となる．

4.4 Syntax(t)

Syntax(t)は高い負の相関となった．また，すべての t において有意な相関が見られた($p<0.01$)．これは，無作為修正者ほどシンタックスエラーを多く起こしていることを示している．Syntax(t)の回数は Score(t)に全く関係していないため，Revs(t)に代わり無作為修正者の検出のためのメトリクスとして活用できることが考えられる．

4.5 Test(t)

Test(t)の相関は解答を開始してから 4 分までは負の相関が得られたが，t=5 以降有意な相関が得られなくなっている．図 1 に t=4 における Test(t)と Score(t)の散布図を示す. Score(t)が 0.4 以下ではテストケースエラーが起こっていない．すなわち，無作為修正者で考えられる Score(t)の低い学生はテストケースエラーをあまり起こさないことが分かる．このため，無作為修正者を検出するためのメトリクスとしての有用性は低いと考える．

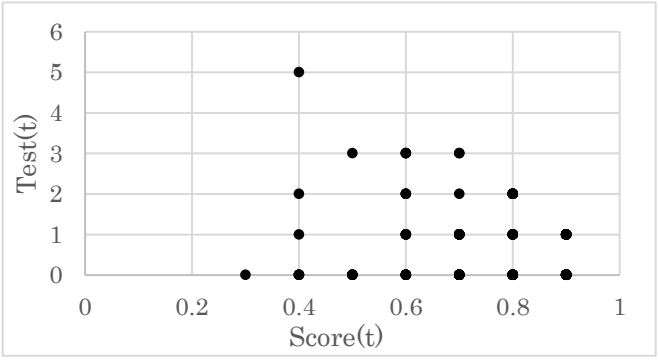

図 1.Test(t)と Score(t)の散布図(t=4)

5. まとめ

本稿では写経型学習において無作為に修正する無作為修正者の検出を目的とし，OJS のスナップショットから検出を可能とするメトリクスを提案し，その有用性を検証した．結果として，先行研究同様 Freq(t)および Revs(t)は Score(t)との有意な相関を得られた．さらに，DiffLine(1,t)においても有意な相関が得られたので，写経型学習での無作為修正者の検出ではこのメトリクスも有用な可能性がある．今後のメトリクスを提案する上での試金石になると考えられる．本研究で新たに提案した Syntax(t)および Test(t)については Revs(t)が Score(t)との依存性があることから，Syntax(t)は Revs(t)の代用として有用だと考えられる．

以上の結果から，写経型学習における無作為修正者は課題型学習における無作為修正者と概ね同様の行動の特徴があると考えられる．これにより，先行研究同様に Freq(t)や Revs(t)を組み合わせることで無作為修正者を検出することが可能だと示唆される．また，Revs(t)を Syntax(t)に変えることや，DiffLine(1,t)も組み合わせることにより写経型学習に特化した検出も可能であることも期待できる．

今後の課題としては，より多くのデータを収集して

分析を行うことにより，結果の精度をさらに高めることが挙げられる．また，複数のメトリクスを組み合わせる手法について提案することが挙げられる．

参 考 文 献

(1) 岡本雅子, 喜多一: “プログラミングの「写経型学習」における初学者のつまずきの類型化とその考察”, 滋賀大学教育学部附属教育実践総合センター紀要, No..22, pp.49-53, (2014)

(2) 岡本雅子, 村上正行, 吉川直人, 喜多, 一:” <実践報告> プログラミングの写経型学習過程を対象としたつまずきの分析とテキスト教材の改善 : 作業の自立的遂行と作業を介した理解のための支援と工夫”, 京都大学高等教育研究, No.19, pp.47-57, (2013)

(3) 大野 優, 上野 秀剛, 内田 眞司:"ソースコードのスナップショットに基づいた無作為修正者の検出", 信学技報 教育工学研究会, Vol.118, No.214, pp.53-58, (2018)

(4) 内田眞司，松村寿枝，西野貴之: ” 反転授業を導入したプログラミング講義の実践と学生の学習履歴分析”，日本高専学会誌（Journal of JACT），Vol.22，No.3，pp.45-48, (2017)

(5) 渡部有隆:”オンラインジャッジの開発と運用: Aizu online judge”, 情報処理,Vol. 56, No. 10, pp. 998, (2015)

(6) 永賢次:”導入プログラミング教育におけるオンラインジャッジシステムの活用の試み”, 情報科学研究, No. 31, pp.25-41, (2011)

(7) 古谷勇樹,長尾和彦,峯脇さやか:”OJS を用いたプログラミング学習支援環境の構築と評価”,電子情報通信学会技術研究報告, Vol. 115, No. 492, pp. 45-50, (2016)